bot_ready

April 20, 2024

0.1 Consuming message from Kafka

For now we are stopping after consuming 1 million messages

[ ]:
```python
from confluent_kafka import Consumer, KafkaError
import json
import time

### Function to fetch the data from the consumer
def process_kafka_message(consumer):
    message = consumer.poll(timeout=0.5)
    if message is None:
        return None
    if message.error():
        if message.error().code() == KafkaError._PARTITION_EOF:
            return None
        else:
            print(f"Error: {message.error()}")
            return None
    try:
        event_data = json.loads(message.value().decode('utf-8'))
        return event_data
    except Exception as e:
        print(f"Error processing message: {e}")
        return None

## Consumer config
consumer_config = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'threat_analytics_consumer_group',
    'auto.offset.reset': 'earliest',
    'enable.auto.commit': False
}
consumer = Consumer(consumer_config)
consumer.subscribe(['threat-analytics-topic'])


event_data_list = []
count = 0  # Count the number of messages consumed
```

```python
### For now we are consuming a million messages and storing it in the array
try:
    while True:
        event_data = process_kafka_message(consumer)
        if event_data is not None:
            event_data_list.append(event_data)
            count += 1
        if count >= 1000000:
            break  # Exit the loop after 50,000 messages
except KeyboardInterrupt:
    pass
finally:
    consumer.close()
```

0.2 Creating Documents

[ ]:
```python
import json
from langchain_text_splitters import RecursiveJsonSplitter

def create_docs(event_data_list):
    json_splitter = RecursiveJsonSplitter(max_chunk_size=1000)
    docs = json_splitter.create_documents(texts=event_data_list)
    return docs

docs = create_docs(event_data_list)
```

0.3 Connecting with Qdrant VectorDB and creating the vector store

Since there is huge amount of data we are first initialising with 1000 elements

Later will perform the batch processing with 10000 elements in one batch

[ ]:
```python
from langchain.vectorstores import Qdrant
from langchain.embeddings import HuggingFaceEmbeddings

url="http://localhost:6333"

doc_store = Qdrant.from_documents(docs[:1000],
                                  HuggingFaceEmbeddings(),
                                  url=url,
                                  grpc_port=6334,
                                  force_recreate=True,
                                  prefer_grpc=True,
                                  collection_name="threat_analytics_vector",
                                 )
```

[ ]:
```
batch_size = 1000  # Define your preferred batch size
for i in range(1000, len(docs), batch_size):
        chunk_batch = docs[i:i + batch_size]
        doc_store.add_documents(chunk_batch)
print("Documents added to Vector Store")
```

0.4 Creating the LLM model using Mistral-7B-Instruct-v0.2

[ ]:
```
from transformers import AutoTokenizer, AutoModelForCausalLM,␣
 ↪BitsAndBytesConfig, pipeline
import torch
from langchain.llms import HuggingFacePipeline


def load_llm():

    #Loading the Mistral Model
    model_name='mistralai/Mistral-7B-Instruct-v0.2'
    tokenizer = AutoTokenizer.from_pretrained(model_name,␣
 ↪trust_remote_code=True)
    tokenizer.pad_token = tokenizer.eos_token
    tokenizer.padding_side = "right"

    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_use_double_quant=True,
        bnb_4bit_compute_dtype=torch.bfloat16
    )

    model = AutoModelForCausalLM.from_pretrained(
        model_name,
        quantization_config=bnb_config,
        )

    text_generation_pipeline = pipeline(
        model=model,
        tokenizer=tokenizer,
        task="text-generation",
        return_full_text=True,
        max_new_tokens=1024,
    )

    llm = HuggingFacePipeline(pipeline=text_generation_pipeline)
    return llm
```

[ ]:
```
llm = load_llm()
```

0.5 Function to answer the query from the User

1. Writing the prompt

2. Creating the rag_chain with StrOutputParser and RunnablePassThrough

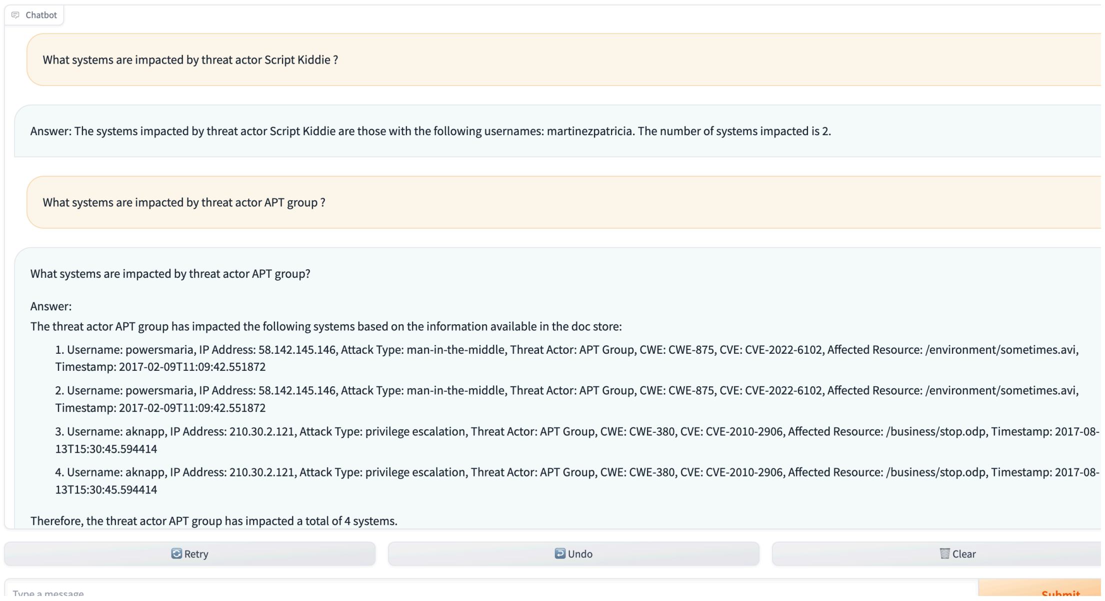
3. Feeding the rendered template to the LLM and parsing the output

[ ]:
```
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough
from langchain_core.output_parsers import StrOutputParser

def answer_query(question, llm, doc_store):
    context_docs = doc_store.similarity_search(question, k= 4)
    context = ' '.join(doc.page_content for doc in context_docs)

    template = f"""You are smart bot. You primarily possess rich expertise in␣
  ↪analysing cybersecurity threats.
Below provided is the content of json that you would get from the store

Context: {context}
Question: {question}


Use the following information to answer the user's question. These are the␣
  ↪system related information of particular system belonging to the user.

default fields:

username: username of the system
ip_address: ip address of the system
user_agent: User Agent is typically the web browser of the system
attack_types: different types of attacks that can happen over the system.␣
  ↪Understand that system has been the victim of this attack
threat_actors: Threat actors are individuals, groups, or organizations that␣
  ↪pose a threat to computer systems, networks, or data and can include a wide␣
  ↪range of entities, such as hackers, cybercriminals, hacktivists,␣
  ↪state-sponsored groups, and insiders
cwe: CWE stands for Common Weakness Enumeration. It is a community-developed␣
  ↪list of software and hardware weakness types that can serve as a common␣
  ↪language for describing software security vulnerabilities.
cve: CVE stands for Common Vulnerabilities and Exposures and Each CVE ID is␣
  ↪associated with a description of the vulnerability, including details such␣
  ↪as affected products, versions, and potential impact.
affected_resource: Its the file that gets affected negatively usually because␣
  ↪of the attack
timestamp: Time at which the attack happened
```

```
If somebody asks if how many systems were impacted by a particular attack or␣
 ↪any other particular field, go through the doc store entirely and calculate␣
 ↪the sum of the systems with that particular field.

A few examples-

Q: What happened with daniel00's system having ip address 113.175.192.202
A: Sure here's what happened - daniel's system was victim of man-in-the-middle␣
 ↪attack and corrupted /own/assume.wav. The threat actor is Script Kiddie with␣
 ↪CWE is CWE-229 and CVE to be CVE-2023-3757.

In case you don't know the answer, just say that you don't know, don't try to␣
 ↪make up an answer. Only return the helpful answer below and nothing else.
"""
    prompt = ChatPromptTemplate.from_template(template)
    chain = (
        {"context": doc_store.as_retriever(search_kwargs={'k': 10}), "question":
 ↪ RunnablePassthrough()}
        | prompt
        | llm
        | StrOutputParser()
    )
    result = llm(template)

    answer = result.replace(template, '')

    return answer
```

0.6 Initialising the Gradio and invoking answer_query function for generating response

[ ]:
```
import time
import gradio as gr

def slow_echo(question, history):
    response = answer_query(question, llm, doc_store)
    for i in range(len(response)):
        time.sleep(0.1)
        #yield "You typed: " + message[: i+1]
        yield response[: i+1]

gr.ChatInterface(slow_echo).launch(share=True)
```



What systems are impacted by threat actor Script Kiddie ?

Answer: The systems impacted by threat actor Script Kiddie are those with the following usernames: martinezpatricia. The number of systems impacted is 2.

What systems are impacted by threat actor APT group ?

What systems are impacted by threat actor APT group?

Answer:
The threat actor APT group has impacted the following systems based on the information available in the doc store:

1. Username: powersmaria, IP Address: 58.142.145.146, Attack Type: man-in-the-middle, Threat Actor: APT Group, CWE: CWE-875, CVE: CVE-2022-6102, Affected Resource: /environment/sometimes.avi, Timestamp: 2017-02-09T11:09:42.551872
2. Username: powersmaria, IP Address: 58.142.145.146, Attack Type: man-in-the-middle, Threat Actor: APT Group, CWE: CWE-875, CVE: CVE-2022-6102, Affected Resource: /environment/sometimes.avi, Timestamp: 2017-02-09T11:09:42.551872
3. Username: aknapp, IP Address: 210.30.2.121, Attack Type: privilege escalation, Threat Actor: APT Group, CWE: CWE-380, CVE: CVE-2010-2906, Affected Resource: /business/stop.odp, Timestamp: 2017-08-13T15:30:45.594414
4. Username: aknapp, IP Address: 210.30.2.121, Attack Type: privilege escalation, Threat Actor: APT Group, CWE: CWE-380, CVE: CVE-2010-2906, Affected Resource: /business/stop.odp, Timestamp: 2017-08-13T15:30:45.594414

Therefore, the threat actor APT group has impacted a total of 4 systems.